What is claimed is.

1. A method for validating a processor design by simulating program execution, comprising the steps of:

identifying a resource that may be accessed by a test program that includes a first simulated process and a second simulated process;

associating a set of non-unique values for said resource;

executing said test program by the steps of:

executing a first sequence of instructions in said first simulated process; and

while performing said step of executing said first sequence, executing a second sequence of instructions in said second simulated process;

wherein said resource is accessed by at least one of said first simulated process and said second simulated process, and wherein upon completion of said steps of executing said first sequence and executing said second sequence, a member of said set of non-unique values is required to be present in said resource; and

verifying an equality between a content of said resource and a member of said set of non-unique values.

2. The method according to claim 1, wherein said resource is a memory resource.

3. The method according to claim 1, wherein said resource is a register.

4. The method according to claim 1, wherein said set of non-unique values is a set of value-lists.

5. The method according to claim 4, wherein said resource comprises a first adjacent resource and a second adjacent resource, and each member of said set of value-lists comprises a first value and a second value, said first value being a permissible value of said first adjacent resource, and said second value being a permissible values of said second adjacent resource, and said step of verifying further comprising the steps of:

identifying a valid member of said set of value-lists, by the steps of:

verifying an equality between a content of said first adjacent resource and said first value of said valid member; and

verifying an equality between a content of said second adjacent resource and said second value of said valid member.

6. The method according to claim 1, wherein said resource comprises a first resource and a second resource and said set of non-unique values comprises a first set of non-unique values that is associated with said first resource, and a second set of non-unique values that is associated with said second resource, further comprising the steps of:

associating a first member of said first set of non-unique values with a second member of said second set of non-unique values; and

wherein said step of verifying comprises the steps of:

verifying an equality between said first resource and said first member; and

verifying an equality between said second resource and said second member.

7. The method according to claim 6, wherein said step of associating said first member is performed by tagging said first member and said second member with a common combination identifier.

8. The method according to claim 6, said first member of said first set of non-unique values comprises a first value-list, and said second member of said second set of non-unique values comprises a second value-list, respective elements of said first value-list being permissible values of said first resource and adjacent resources thereof, and respective elements of said second value-list being permissible values of said second resource and adjacent resources thereof,

wherein said step of verifying comprises the steps of

verifying an equality between a content of said first resource and adjacent resources thereof with corresponding elements of said first value-list; and

verifying an equality between a content of said second resource and adjacent resources thereof with corresponding elements of said second value-list.

9. The method according to claim 1, wherein said step of associating said set of non-unique values is performed prior to said step of executing said first sequence of instructions.

10. The method according to claim 9, further comprising the steps of:

defining a results section in said test program, and

entering all permissible values assumable by said resource and an identifier of said resource in an entry of said results section.

11. The method according to claim 1, wherein said step of executing said test program further comprises the steps of:

generating said first sequence and said second sequence to define generated instructions;

while performing said step of generating, simulating one of said generated instructions in said first simulated process and said second simulated process;

maintaining a store that contains a set of values that are assumable in said resource during said step of simulating said one of said generated instructions; and

thereafter determining whether said store contains non-unique values.

12. The method according to claim 11, further comprising the steps of:

storing in said store first values of said resource during accesses thereof by said first simulated process; and

storing in said store second values of said resource by said second simulated process during accesses thereof, wherein said first values comprise first written values, and said second values comprise second written values; and;

identifying in said store a last value written to said resource in said step of simulating said one of said generated instructions.

13. The method according to claim 11,

wherein said step of maintaining said store further comprises the steps of:

determining an initial state of said test program immediately prior to performing said step of simulating said one of said generated instructions;

storing in said store written values of target resources of said one generated instruction during accesses thereof; and

further comprising the steps of:

identifying all combinations of values of input resources of said one generated instruction;

rolling back said test program to reestablish said initial state;

thereafter resimulating said one of said generated instructions using each of said combinations of values;

reidentifying written values that are written to said target resouruces; and

updating said store with reidentified written values of said target resources.

14. The method according to claim 1, further comprising the step of establishing a synchronization barrier for said first simulated process and said second simulated process.

15. The method according to claim 1, further comprising the steps of biasing generation of said test program to promote collisions of memory accessing instructions that are executed by said first simulated process and said second simulated process.

16. A method of verification of an architecture by simulation, comprising the steps of:

defining a program input to a test generator;

generating a test program responsive to said program input said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results, wherein at least one member of said list of predicted resource results comprises a plurality of permissible results;

simulating an execution of said test program using a plurality of simultaneously executing processes; and

verifying an actual resource result by determining that at least one of said plurality of permissible results is equal to said actual resource result.

17. The method according to claim 16, wherein said list of predicted resource results comprises predicted results of adjacent resources, wherein said adjacent resources are mutually dependent, and said step of verifying said actual resource result is performed by verifying each of said adjacent resources.

18. The method according to claim 17, wherein said adjacent resources are memory resources.

19. The method according to claim 17, wherein said adjacent resources are registers.

20. The method according to claim 16, wherein said list of predicted resource results comprises predicted results of mutually dependent non-adjacent resources, further comprising the steps of:

identifying a combination of said mutually dependent non-adjacent resources by tagging corresponding members of said list of predicted resource results with a unique combination identifier to define commonly tagged value-lists of predicted resource results; and

said step of verifying said actual resource result is performed by verifying that resources of said combination have actual results that are equal to a member of a corresponding one of said commonly tagged value-lists.

21. A method of predicting non-unique results by simulating a system design, comprising the steps of:

defining a program input to a test generator;

generating a test program responsive to said program input, said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results, wherein at least one member

of said list of predicted resource results comprises a plurality of permissible results;

simulating an execution of a single instruction of said test program by a first process of said test program; and

calculating possible values of target resources of said single instruction.

22. The method according to claim 21, wherein said target resources are memory resources.

23. The method according to claim 21, wherein said target resources are registers.

24. The method according to claim 21, further comprising the steps of:

performing said step of simulating an execution by a second process of said test program;

maintaining process-linked lists of written values that are written to said target resources of said single instruction by said first process and said second process; and

determining respective last values in said process-linked lists of written values.

25. The method according to claim 16, further comprising the step of establishing a synchronization barrier for said simultaneously executing processes.

26. The method according to claim 16, further comprising the steps of biasing generation of said test program to promote collisions of memory accessing instructions that are executed by said simultaneously executing processes.

27. A computer software product, comprising a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to execute a method for validating a processor design by simulating program execution, comprising the steps of:

identifying a resource that may be accessed by a test program that includes a first simulated process and a second simulated process;

associating a set of non-unique values for said resource;

executing said test program by the steps of:

executing a first sequence of instructions in said first simulated process; and

while performing said step of executing said first sequence, executing a second sequence of instructions in said second simulated process;

wherein said resource is accessed by at least one of said first simulated process and said second simulated process, and wherein upon completion of said steps of executing said first sequence and executing said second

sequence, a member of said set of non-unique values is required to be present in said resource; and

verifying an equality between a content of said resource and a member of said set of non-unique values.

28. The computer software product according to claim 27, wherein said resource is a memory resource.

29. The computer software product according to claim 27, wherein said resource is a register.

30. The computer software product according to claim 27, wherein said set of non-unique values is a set of value-lists.

31. The computer software product according to claim 30, wherein said resource comprises a first adjacent resource and a second adjacent resource, and each member of said set of value-lists comprises a first value and a second value, said first value being a permissible value of said first adjacent resource, and said second value being a permissible values of said second adjacent resource, and said step of verifying further comprising the steps of:

identifying a valid member of said set of value-lists, by the steps of:

verifying an equality between a content of said first adjacent resource and said first value of said valid member; and

verifying an equality between a content of said second adjacent resource and said second value of said valid member.

32. The computer software product according to claim 27, wherein said resource comprises a first resource and a second resource and said set of non-unique values comprises a first set of non-unique values that is associated with said first resource, and a second set of non-unique values that is associated with said second resource, the method further comprising the steps of:

associating a first member of said first set of non-unique values with a second member of said second set of non-unique values; and

wherein said step of verifying comprises the steps of:

verifying an equality between said first resource and said first member; and

verifying an equality between said second resource and said second member.

33. The computer software product according to claim 32, wherein said step of associating said first member is performed by tagging said first member and said second member with a common combination identifier.

34. The computer software product according to claim 32, said first member of said first set of non-unique values comprises a first value-list, and said second member of said second set of non-unique values comprises a second value-list, respective elements of said first value-list being permissible values of said first resource and adjacent resources thereof, and respective elements of said second value-list being permissible values of said second resource and adjacent resources thereof,

wherein said step of verifying comprises the steps of

verifying an equality between a content of said first resource and adjacent resources thereof with corresponding elements of said first value-list; and

verifying an equality between a content of said second resource and adjacent resources thereof with corresponding elements of said second value-list.

35. The computer software product according to claim 27, wherein said step of associating said set of non-unique values is performed prior to said step of executing said first sequence of instructions.

36. The computer software product according to claim 35, wherein said step of associating said set of non-unique values is performed by the steps of:

defining a results section in said input statements, and

entering all permissible values assumable by said resource and an identifier of said resource in an entry of said results section.

37. The computer software product according to claim 27, wherein said step of executing said test program further comprises the steps of:

generating said first sequence and said second sequence to define generated instructions;

while performing said step of generating, simulating one of said generated instructions in said first simulated process and said second simulated process;

maintaining a store that contains a set of values that are assumable in said resource during said step of simulating said one of said generated instructions; and

thereafter determining whether said store contains non-unique values.

38. The computer software product according to claim 37, wherein said step of maintaining said store further comprises the steps of:

maintaining a first store that contains first values contained in said resource during accesses thereof by said first simulated process; and

maintaining a second store that contains second values contained in said resource by said second simulated

process during accesses thereof, wherein said first values comprise first read values and first written values, and said second values comprise second read values and second written values; and

said step of determining whether said store contains non-unique values further comprises the step of identifying in one of said first store and said second store a last value written to said resource in said step of simulating said one of said generated instructions.

39. The computer software product according to claim 38, the method further comprising the steps of:

identifying a first unique combination of said first simulated process, and said first read values;

identifying a second unique combination of said second simulated process, and said second read values; and

resimulating said one of said generated instructions using a selected value that is selected from said first read values of said first unique combination and said second read values of said second unique combination as an input to said one of said generated instructions; and

reidentifying values that are written to said resource in said step of resimulating.

40. The computer software product according to claim 27, further comprising the step of establishing a synchronization barrier for said first simulated process and said second simulated process.

41. The computer software product according to claim 27, further comprising the step of biasing generation of said test program to promote collisions of memory accessing instructions that are executed by said first simulated process and said second simulated process.

42. A computer software product, comprising a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to perform a method of verification of an architecture by simulation, comprising the steps of:

defining a program input to a test generator;

generating a test program responsive to said program input, said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results, wherein at least one member of said list of predicted resource results comprises a plurality of permissible results;

simulating an execution of said test program using a plurality of simultaneously executing processes; and

verifying an actual resource result by determining that at least one of said plurality of permissible results is equal to said actual resource result.

43. The computer software product according to claim 42, wherein said list of predicted resource results comprises predicted results of adjacent resources, wherein said adjacent resources are mutually dependent, and said step of verifying said actual resource result is performed by verifying each of said adjacent resources.

44. The computer software product according to claim 43, wherein said adjacent resources are memory resources.

45. The computer software product according to claim 43, wherein said adjacent resources are registers.

46. The computer software product according to claim 42, further comprising the step of establishing a synchronization barrier for said simultaneously executing processes.

47. The computer software product according to claim 42, further comprising the steps of biasing generation of said test program to promote collisions of memory accessing instructions that are executed by said simultaneously executing processes.

48. The computer software product according to claim 42, wherein said list of predicted resource results comprises predicted results of mutually dependent non-adjacent resources, the method further comprising the steps of:

identifying a combination of said mutually dependent non-adjacent resources by tagging corresponding members of said list of predicted resource results with a unique combination identifier to define commonly tagged value-lists of predicted resource results; and

said step of verifying said actual resource result is performed by verifying that resources of said combination have actual results that are equal to a member of a corresponding one of said commonly tagged value-lists.

49. A computer software product, comprising a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to perform a method of predicting non-unique results by simulating a system design, comprising the steps of:

defining a program input to a test generator;

generating a test program responsive to said program input, said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results, wherein at least one member of said list of predicted resource results comprises a plurality of permissible results;

simulating an execution of a single instruction of said test program by a first process of said test program; and

calculating possible values of target resources of said single instruction.

50. The computer software product according to claim 49, the method further comprising the steps of:

performing said step of simulating an execution by a second process of said test program;

maintaining process-linked lists of written values that are written to said target resources of said single instruction by said first process and said second process; and

determining respective last values in said process-linked lists of written values.

51. The computer software product according to claim 50, the method further comprising the steps of:

prior to performing said steps of simulating said execution by said first process and of simulating said execution by said second process memorizing an initial simulated state of said test program;

maintaining process-linked lists of read values that are read from source resources of said single instruction;

identifying a member of said lists of read values having a unique value to define an identified member;

restoring said initial simulated state of said test program; and

performing said step of simulating said execution a second time, reading said identified member, using an associated process thereof.

52. An apparatus for verifying a design comprising a simulation processor which is adapted to perform the steps of:

identifying a resource that may be accessed by a test program that includes a first simulated process and a second simulated process;

associating a set of non-unique values for said resource;

executing said test program by the steps of:

executing a first sequence of instructions in said first simulated process; and

while performing said step of executing said first sequence, executing a second sequence of instructions in said second simulated process;

wherein said resource is accessed by at least one of said first simulated process and said second simulated process, and wherein upon completion of said steps of executing said first sequence and executing said second sequence, a member of said set of non-unique values is required to be present in said resource; and

verifying an equality between a content of said resource and a member of said set of non-unique values.

53. The apparatus according to claim 52, wherein said resource is a memory resource.

54. The apparatus according to claim 52, wherein said resource is a register.

55. The apparatus according to claim 52, wherein said resource comprises a first adjacent resource and a second adjacent resource, and said set of non-unique values comprises a first subset of non-unique values and a second subset of non-unique values, said first subset of non-unique values being permissible values of said first adjacent resource, and said second subset of non-unique values being permissible values of said second adjacent resource, wherein said processor is further adapted to perform the steps of:

establishing at least one allowable subcombination of members of said set of non-unique values, a first member of said subcombination being selected from said first subset of non-unique values and a second member of said subcombination being selected from said second subset of non-unique values; and

wherein said step of verifying is performed by the steps of:

verifying an equality between a content of said first adjacent resource and said first member of said subcombination; and

verifying an equality between a content of said second adjacent resource and said second member of said sub-combination.

56. The apparatus according to claim 52, wherein said resource comprises a first resource and a second resource and said set of non-unique values comprises a first set of non-unique values that is associated with said first resource, and a second set of non-unique values that is associated with said second resource, wherein said processor is further adapted to perform the steps of:

associating a first member of said first set of non-unique values with a second member of said second set of non-unique values; and

wherein said step of verifying comprises the steps of:

verifying an equality between said first resource and said first member; and

verifying an equality between said second resource and said second member.

57. The apparatus according to claim 56, wherein said step of associating said first member is performed by tagging said first member and said second member with a common combination identifier.

58. The apparatus according to claim 56, said first member of said first set of non-unique values comprises a first value-list, and said second member of said second set of non-unique values comprises a second value-list, respective elements of said first value-list being permissible values of said first resource and adjacent resources thereof, and respective elements of said second value-list being permissible values of said second resource and adjacent resources thereof,

wherein said step of verifying comprises the steps of

verifying an equality between a content of said first resource and adjacent resources thereof with corresponding elements of said first value-list; and

verifying an equality between a content of said second resource and adjacent resources thereof with corresponding elements of said second value-list.

59. The apparatus according to claim 52, wherein said step of associating said set of non-unique values is performed prior to said step of executing said first sequence of instructions.

60. The apparatus according to claim 59, wherein said step of associating said set of non-unique values is performed by the steps of:

providing input statements to a test generator; and

defining a results section in said input statements, and

entering all permissible values assumable by said resource and an identifier of said resource in an entry of said results section.

61. The apparatus according to claim 52, wherein said step of executing said test program further comprises the steps of:

generating said first sequence and said second sequence to define generated instructions;

while performing said step of generating, simulating one of said generated instructions in said first simulated process and said second simulated process;

maintaining a store that contains a set of values that are assumable in said resource during said step of simulating said one of said generated instructions; and

thereafter determining whether said store contains non-unique values.

62. The apparatus according to claim 61, wherein said step of maintaining said store further comprises the steps of:

maintaining a first store that contains first values contained in said resource during accesses thereof by said first simulated process; and

maintaining a second store that contains second values contained in said resource by said second simulated process during accesses thereof, wherein said first values comprise first read values and first written values,

and said second values comprise second read values and second written values; and

said step of determining whether said store contains non-unique values further comprises the step of identifying in one of said first store and said second store a last value written to said resource in said step of simulating said one of said generated instructions.

63. The apparatus according to claim 62, wherein said processor is further adapted to perform the steps of:

identifying a first unique combination of said first simulated process, and said first read values;

identifying a second unique combination of said second simulated process, and said second read values; and

resimulating said one of said generated instructions using a selected value that is selected from said first read values of said first unique combination and said second read values of said second unique combination as an input to said one of said generated instructions; and

reidentifying values that are written to said resource in said step of resimulating.

64. The apparatus according to claim 63, wherein said processor is further adapted to perform the steps of determining an initial state of said test program immediately prior to performing said step of simulating said one of said generated instructions; and

rolling back said test program to reestablish said initial state prior to performing said step of resimulating.

65. An apparatus for verifying an architecture by simulation comprising a simulation processor which is adapted to perform the steps of:

defining a program input to a test generator;

generating a test program responsive to said program input, said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results, wherein at least one member of said list of predicted resource results comprises a plurality of permissible results;

simulating an execution of said test program using a plurality of simultaneously executing processes; and

verifying an actual resource result by determining that at least one of said plurality of permissible results is equal to said actual resource result.

66. The apparatus according to claim 65, wherein said list of predicted resource results comprises predicted results of adjacent resources, wherein said adjacent resources are mutually dependent, and said step of verifying said actual resource result is performed by verifying each of said adjacent resources.

67. The apparatus according to claim 65, wherein said list of predicted resource results comprises predicted results of mutually dependent non-adjacent resources, wherein said processor is further adapted to perform the steps of:

identifying a combination of said mutually dependent non-adjacent resources by tagging corresponding members of said list of predicted resource results with a unique combination identifier to define commonly tagged value-lists of predicted resource results; and

said step of verifying said actual resource result is performed by verifying that resources of said combination have actual results that are equal to a member of a corresponding one of said commonly tagged value-lists.

68. An apparatus for design verification in which program instructions are stored, which instructions cause a processor to execute a method of predicting non-unique results by simulating a system design, comprising the steps of:

defining a program input to a test generator;

generating a test program responsive to said program input, said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results, wherein at least one member of said list of predicted resource results comprises a plurality of permissible results;

simulating an execution of a single instruction of said test program by a first process of said test program; and

calculating possible values of target resources of said single instruction.

69. The apparatus according to claim 68, wherein said instructions cause the processor to further perform the steps of:

performing said step of simulating an execution by a second process of said test program;

maintaining process-linked lists of written values that are written to said target resources of said single instruction by said first process and said second process; and

determining respective last values in said process-linked lists of written values.

70. The apparatus according to claim 69, wherein said instructions cause the processor to further perform the steps of:

prior to performing said steps of simulating said execution by said first process and of simulating said execution by said second process memorizing an initial simulated state of said test program;

maintaining process-linked lists of read values that are read from source resources of said single instruction;

identifying a member of said lists of read values having a unique value to define an identified member;

restoring said initial simulated state of said test program; and

performing said step of simulating said execution a second time, reading said identified member, using an associated process thereof.